E2PROM 数据手册 E2PROM V 1.7

资源	PSoC® 模块			API 内存 （字节）		引脚 （根据外部 I/O）
	数字	模拟 CT	模拟 SC	闪存	RAM	
CY8C29x66, CY8CLED16, CY8CPLC20, CY8CLED16P01	0	0	0	26+1013 + 大小	请参见下面的内容	0
CY8C27x43、CY8C24x94、CY8C22x13、CY7C64215、CY8CLED04/08、CY8CLED0xD、CY8CLED0xG、CY8C22x45、CY8C28x45、 CY8C28xxx	0	0	0	25+750 + 大小	请参见下面的内容	0
CY8C24x23A、CY8C23x33、CY8C21x23、 CY8CLED02	0	0	0	25+750 + 大小	请参见下面的内容	0
CY8C21x34、CY7C603xx、CYWUSB6953、CY8C20x24、CY8C20x34、CY8C21x45、CY8C21x12	0	0	0	25+1044 + 大小	请参见下面的内容	0
CY8C26/25xxx	0	0	0	19+777 + 大小	请参见下面的内容	0

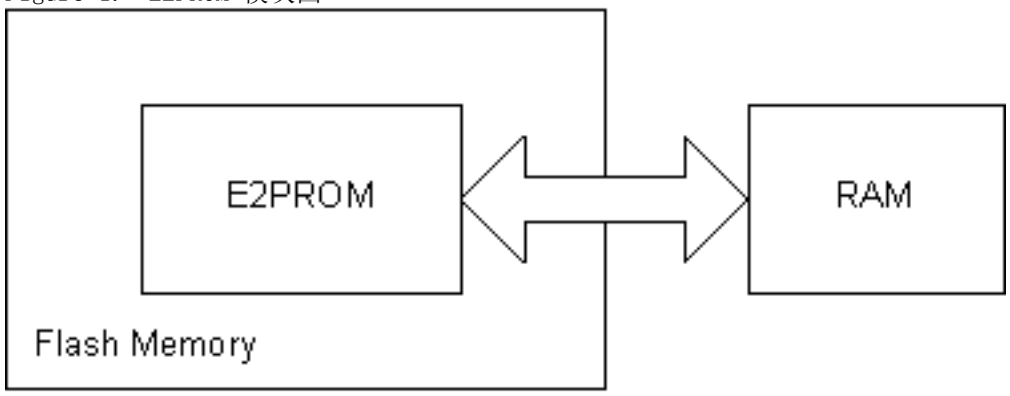
有关使用此用户模块的一个或多个完整配置的实用型示例项目，请转到 www.cypress.com/psocexampleprojects.

功能和概述

- 完全面向字节的 EEPROM 仿真
- 抽象的面向区块的闪存结构
- 高效使用存储器

EEPROM 用户模块模拟 PSoC 设备闪存中的 EEPROM 设备。 EEPROM 设备可以定义为起始于闪存区块的任意边界，字节长度为 1 到闪存空间的剩余大小。 API 使用户能够一次读取和写入 1 到 N 个字节。

Figure 1. EEPROM 模块图



功能说明

EEPROM 用户模块是一种软件算法，它不使用 PSoC 设备的硬件资源。 可以创建这些 EEPROM 虚拟设备的一个或多个实例。

闪存根据设备进行组织，对于 32K 设备，组织为 511 个大小为 64 字节的区块；对于 16K 设备，组织为 256 个大小为 64 字节的区块；对于 8K 设备，组织为 128 个大小为 64 字节的区块；对于 4K 设备，组织为 64 个大小为 64 字节的区块。 PSoC 的结构允许逐字节读取闪存数据，但是需要逐区块写入数据 – 一次写入 64 字节。 此用户模块的目的是在基于闪存的存储设备 （面向字节读取、区块写入的设备）上模拟 EEPROM 设备 （面向字节读取、字节写入的设备）。

EEPROM 存储区域开始于闪存区块边界，由 1 个或多个字节组成。 可以使用 E2Read() 和 E2Write() API 子程序访问此虚拟设备。 虚拟地址空间范围为从 0 到 N-1，其中 N 是 EEPROM 设备的长度 / 大小 （以字节表示）。 在闪存使用中还必须考虑保留的存储区域大小，但是此大小不能通过资源计来计算。 在上面的存储器使用表中，此大小元素标识为 “ 大小 ”。

E2Read() API 算法使用 ROMX M8C 指令逐字节高效读取闪存。 此算法需要使用 RAM 的最后 8 个字节：0xF8 到 0xFF。

E2Write() API 算法将数据逐区块写入闪存。 根据 EEPROM 存储空间的起始地址偏移，E2Write() 对要写入到区块边界上对齐的段的数据进行解析。 此算法也需要使用 RAM 的最后 8 个字节： 0xF8 到 0xFF。

对于跨整个区块的大小为 64 字节的段，将调用 FlashBlockWrite() 子程序，然后写入区块。

对于小于 64 字节的段，临时 64-byte 堆栈缓冲区由未写入的数据和经过修改的数据组成。 在写入数据后，将从堆栈释放缓冲区。 这是将数据保存在区块中时所必需的操作。

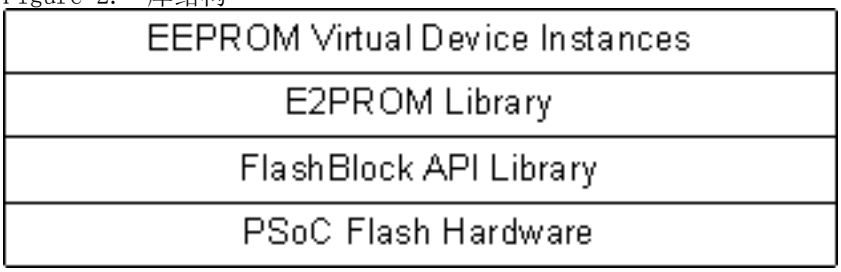
要通过优化参数降低系统开销，请参见上面的*参数和资源*一节。

软件组织

下图显示如何将软件组织到一些由两个库和 N 个 EEPROM 虚拟设备实例组成的层中。

上述资源使用列出了闪存中的 API 库大小以及 FlashBlock API 库和 E2PROM 库中包含的附加代码。 如果使用此设备的多个实例，则资源计中仅计算 UserModule E2PROM API 长度。 对于任意数量的用户模块实例，仅使用一个 FlashBlock API 库和 E2PROM 库。

Figure 2. 库结构



FlashBlock API 库提供用于启用基本闪存区块读写子程序的 API。 这些子程序直接连接到硬件，使设备可以进入系统监控模式。 在此期间，将屏蔽所有中断。 对于实例化的 EEPROM 虚拟设备的所有实例，仅需要链接一次此库。

E2PROM 库通过区块对齐和缓冲，将区块操作转换为按字节操作。 然后，它调用 FlashBlock API 库函数以写入闪存。 无论 EEPROM 虚拟设备的数量如何，仅需要链接一次此库。

EEPROM 虚拟设备层提供一个允许存在 EEPROM 设备多个实例的简单方法，而开销只是 E2PROM 库和 FlashBlock API 库一个实例的开销。 EEPROM 虚拟设备的每个实例包含此源代码的自定义副本，此源代码的长度最小，只有 16 字节。

E2PROM 和 FlashBlock API 库都包括在库文件中。 如果调用这些函数，则此文件自动链接到项目中。

Note 不再重新进入这些库。

闪存写入周期寿命

闪存对生命周期写入总数有限制。 可以在相应的设备系列数据手册中找到此设备特定的总数。

闪存保护

必须编辑 *flashsecurity.txt* 文件，闪存才能写入组成 EEPROM 设备的模块。

flashsecurity.txt 中有一个针对设备各闪存区块的 ASCII 字符。 此 ASCII 字符的有效选项为“W”、“R”、“U”和“F”。 此文件中的 ASCII 字符定义了要应用到每个闪存区块的闪存安全设置。 “W”是默认设置，它完全阻止写入闪存区块。 要允许 E2PROM 用户模块读写区块，保护选项应更改为“R”、“F”或“U”。 应当为 E2PROM 占据的每个闪存区块更改该选项。 要正确更改 *flashsecurity.txt* 中的安全设置，请删除受影响的闪存区块的对应字符。 键入与所需的安全设置对应的新字符。 保存项目，然后重建项目，以确保更改的设置生效。

模式	设置	说明	在 PSoC Designer 中
00b	SR ER EW IW	无保护	U = 无保护
01b	SR ER EW IW	读取保护	F = 工厂升级
10b	SR ER EW IW	禁用外部写入	R = 字段升级
11b	SR ER EW IW	禁用内部写入	W = 完全保护

存储器的高效使用

可以使用下面的指南最大程度地提高 RAM 和闪存资源的效率。

如果闪存非常“充足”（高效使用 RAM）

- 即使要存储的数据较少，也应以 64-byte 长度的倍数创建 EEPROM 设备。
- 始终以 64-byte 长度的倍数写入数据。
- 始终以正确的字节数读取数据。

例如，序列号数据的长度为 10 字节。 用名称序列号创建 EEPROM 用户模块。 将长度参数设置为 64。 这将在闪存中分配 64-byte 区块。 在 RAM 中，设置 10-byte 数据阵列以保存序列号数据。 当通过 E2Write() 写入序列号时，请将 wByteCount 参数指定为 64。 但是，当读取序列号数据时，请将 wByteCount 参数指定为 10。

如果闪存空间是主要考虑（高效使用闪存）

- 仅使用所需的字节数创建 EEPROM 设备。
- 始终以正确的字节数写入数据。
- 始终以正确的字节数读取数据。

定时

下表汇总了 EEPROM 算法的定时：

Table 1. 除非下表中另有指定，否则保证的所有限制值均针对以下条件而言：T_A = 25° C，V_{dd} = 5.0V。

参数	条件和注释	典型值	限制	单位
写入时间 (0° C-100° C)	写入时间取决于需要更新的闪存区块数。[1,2]	45	120[3,4]	毫秒 / 区块
写入时间 (-40° C-0° C)	写入时间取决于需要更新的闪存区块数。[1,2]	70	240[3,4]	毫秒 / 区块
读取时间	81 + 46N 个时钟周期	--	--	时钟周期 / 字节
	最坏的情况为 1 字节：	--	127	
	对于 16 字节： (81+46(16))/10	51	--	
	对于 64 字节： (81+46(64))/64	48	--	

定时表说明

1. 如果 M8C 处于系统监控模式，则在闪存的读取和写入操作期间将屏蔽中断。
2. 写入时间由擦除并写入每个闪存区块的 FlashBlockWrite() 子程序确定。
3. 闪存写入总数是有限制的。 有关每个区块的写入周期数的规定，请参见部件特定的数据手册。
4. 限制是针对正常编程操作指定的。 在使用调试器的仿真模式中，写入脉冲可能耗费 1 到 2 秒。

直流和交流电气特性

有关闪存特性和规范，请参见 PSoC 器件的器件数据手册。

放置

EEPROM 用户模块是在软件中实现的，不需要放置。

参数和资源

FirstBlock

FirstBlock 是 EEPROM 器件驻留的起始模块。 值的范围为 0 到特定器件的最大闪存模块数： 对于 32K 器件，为 511；对于 16K 器件，为 255；对于 8K 器件，为 127；对于 4K 器件，为 63；对于 2K 器件，为 31。

必须注意确保 E2PROM 用户模块的闪存位置不要覆盖闪存中的任何代码或其他重要数据。 应当将 E2PROM 存储器定位在闪存中尽可能高的位置。 例如，如果器件有 512 个闪存模块，将 E2PROM 用户模块定位在闪存模块 511 是一个不错的选择。 将其定位在闪存模块 0 或 1 可能导致系统故障，这是因为 E2PROM 位置会覆盖中断矢量。 任何在模块 0 或 1 向 E2PROM 进行的写入操作都会覆盖中断矢量表。 将 E2PROM 定位在闪存中央附近也不是好的做法。 ImageCraft 编译器不会为 E2PROM 保留单独的存储器区域。 因此，随着代码越来越大，它会逐渐覆盖闪存中央附近的 E2PROM 位置。

长度

长度定义以字节为单位的 EEPROM 器件大小。 有效范围为 1 到 N，其中 N 是以字节为单位的 EEPROM 虚拟器件的大小。

警告： 确保虚拟 EEPROM 器件的物理位置不超过器件的闪存大小。 EEPROM 器件的第一个物理字节位于 FirstBlock * 64。 该虚拟器件的最后一个字节物理上位于 FirstBlock * 64 + Length - 1。 不执行错误检查。 如果无法确保不超过该大小，则可能导致闪存不可预见地写入不需要的模块中。

RAM 存储器开销

将使用下面的 RAM 存储器资源：

API 子程序	存储器使用
E2Read()	堆栈空间： 8 字节 RAM 高存储器： 0xF8 - 0xFF
不带部分模块写入的 E2Write()	堆栈空间： 32 字节 RAM 高存储器： 9 字节 0xF8 - 0xFF
带部分模块写入的 E2Write()	堆栈空间： 103 字节 RAM 高存储器： 9 字节 0xF8 - 0xFF

应用程序编程接口

应用程序编程接口 (API) 子程序作为用户模块的一部分提供，从而使设计人员能够采用更高级的方式处理模块。 本节指定每个函数的接口，以及 “ 包含 ” 文件所提供的相关常量。

Note 此处就像在所有用户模块 API 中一样，可以通过调用 API 函数更改 A 和 X 寄存器的值。 如果在调用后需要 A 和 X 的值，则调用函数负责在调用前保留 A 和 X 的值。 此 “ 寄存器易失 ” 策略是针对提高效率的目的选择，自 PSoC Designer 1.0 版起已强制使用此策略。 C 编译器自动遵循此要求。 汇编语言程序员也必须确保其代码遵守这一策略。 虽然一些用户模块 API 函数可以保留 A 和 X 不变，但是无法保证它们将来也会如此。

下面将介绍 EEPROM 用户模块 API。

E2PROM_Start

说明：

一个空函数，维护它的目的是为了确保用户模块 API 一致性。

C 原型：

```
void  E2PROM_Start(void)
```

汇编程序：

```
lcall  E2PROM_Start
```

参数：

无

返回值：

无

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在 CY8C29xxx 中的大内存模式下，所有 RAM 页指针寄存器也会出现这种状况。 如果需要，由调用函数负责将调用前后的数值保存在 fastcall16 函数内。

E2PROM_Stop

说明：

一个空函数，维护它的目的是为了确保用户模块 API 一致性。

C 原型：

```
void  E2PROM_Stop(void)
```

汇编程序：

```
lcall  E2PROM_Stop
```

参数：

无

返回值：

无

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在 CY8C29xxx 中的大内存模式下，所有 RAM 页指针寄存器也会出现这种状况。 如果需要，由调用函数负责将调用前后的数值保存在 fastcall16 函数内。

E2PROM_bE2Write

说明：

将指定的数据从 RAM 缓冲区写入定义的 EEPROM。 请注意，必须正确设置 *flashsecurity.txt* 文件才允许写入闪存。

C 原型：

```
CHAR  E2PROM_bE2Write(WORD wAddr, BYTE *pbData, WORD wByteCount,
                      CHAR cTemperature);
```

汇编：

```
push  X
mov   X, SP
mov   A, <cTemperature>               ; cTemperature argument
push  A
mov   A, <wByteCount>                 ; wByteCount - MSB
push  A
mov   A, <wByteCount+1>               ; wByteCount - LSB
push  A
mov   A, <pbData>                     ; pbData - MSB
push  A
mov   A, <pbData+1>                   ; pbData - LSB
push  A
mov   A, <wAddr>                      ; wAddr - MSB
push  A
mov   A, <wAddr+1>                    ; wAddr - LSB
push  A
lcall  E2PROM_bE2Write
add   SP, -E2_WR_ARG_STACK_FRAME_SIZE ; restore call stack
pop   X
```

其中 <..> 指的是用于将参考数据放入累加器的任何寻址模式或指令数。

参数：

wAddr： 从其写入 RAM 数据的 EEPROM 器件地址空间的地址偏移。 它可以为 0 到 N-1，其中 N 是 EEPROM 器件的长度。

pbData： 指向包含要写入数据的 RAM 缓冲区的指针。

wByteCounter： 要写入闪存的字节数。

cTemperature： 以摄氏度表示的 PSoC die 温度。 可以使用下列选项之一指定此值：

- 用户模块，例如 FlashTemp。
- 外部器件或传感器。
- 适用于 PSoC 器件经历的所有环境条件的额定值。

  例如，室温 = 25° C。

返回值：

可以返回下列值：

返回标志	说明	值
NOERROR	成功完成操作。	0
FAILURE	未成功完成操作。 最可能是闪存保护位错误所致。	-1
STACKOVERFLOW	堆栈空间不足，难以满足算法要求。	-2

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在 CY8C29xxx 中的大内存模式下，所有 RAM 页指针寄存器也会出现这种状况。 如果需要，由调用函数负责将调用前后的数值保存在 fastcall16 函数内。 当前，仅修改 CUR_PP、IDX_PP 和 MVW_PP 页指针寄存器。

E2PROM_E2Read

说明：

将指定的 EEPROM 器件数据从闪存读取到指定的 RAM 缓冲区中。

C 原型：

```
void E2PROM_E2Read(WORD wAddr, BYTE *pbData, WORD wByteCount)
```

汇编程序：

```
push  X
mov   X, SP
mov   A, <wByteCount>                  ; wByteCount - MSB
push  A
mov   A, <wByteCount+1>                ; wByteCount - LSB
push  A
mov   A, <pbDataDest>                  ; pbDataDest - MSB
push  A
mov   A, <pbDataDest+1>                ; pbDataDest - LSB
push  A
mov   A, <wAddr>                       ; wAddr - MSB
push  A
mov   A, <wAddr+1>                     ; wAddr - LSB
push  A
lcall   E2PROM_E2Read
add   SP, -E2_RD_ARG_STACK_FRAME_SIZE  ; restore call stack
pop   X
```

其中 <..> 指的是用于将参考数据放入累加器的任何寻址模式或指令数。

参数：

wAddr： 从中读取闪存数据的 EEPROM 器件地址空间的地址偏移。 它可以为 0 到 N-1，其中 N 是 EEPROM 器件的长度。

pbData： 指向数据读入的 RAM 缓冲区的指针。

wByteCounter： 要从闪存读取的字节数。

返回值：

无

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在 CY8C29xxx 中的大内存模式下，所有 RAM 页指针寄存器也会出现这种状况。 如果需要，由调用函数负责将调用前后的数值保存在 fastcall16 函数内。 当前，仅修改 CUR_PP 和 MVW_PP 页指针寄存器。

固件源代码示例

下面是 C 语言源代码。

```
//*****************************************************************************
//*    EEPROM User Module Example Code:
//*
//*    A SerialNumber EEPROM was created to start at block 250 with a length
//*    of 10 bytes. Remember to edit the Flashsecurity.txt file to allow
//*    writes to flash  (Set the 250th block to U - Unprotected).
//*
//*    This example:
//*
//*        a) Writes the initial data to the EEPROM block area
//* Note that this will invoke the SavePartial algorithm which
//* allocates a temporary 64-byte buffer on the stack. If
//* Flash memory is plentiful and the extra 54 bytes can be
//* wasted, set the SerialNumber device to a length of 64 and when
//* writing, specify a bytecount of 64. This will write an entire
//* block without using a temporary buffer. The extra 54 bytes
//* beyond the SerialNumber will be bogus data.
//*
//*        b) Reads the data back into a RAM buffer
//*
//*****************************************************************************/

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

#define  ADDRESS_OFFSET    0
#define  NUMBER_OF_BYTES   10
#define  TEMPERATURE       25

/* Initialize a RAM buffer with default Serial Number */
BYTE  abInitialSerialNumber[] = {'0','1','2','3','4','5','6','7','8','9' };
BYTE  abSerialNumberBuffer[NUMBER_OF_BYTES];

void main(void)
{
    BYTE bError;


    /* Write the Serial Number - assume temp of 25C */
    bError = E2PROM_bE2Write(ADDRESS_OFFSET, abInitialSerialNumber, NUMBER_OF_BYTES,
TEMPERATURE);


    if ( bError == E2PROM_NOERROR )
    {
        /* Read the Serial Number back into a RAM buffer */
```

```
        E2PROM_E2Read( ADDRESS_OFFSET, abSerialNumberBuffer, NUMBER_OF_BYTES );
    }

    while(1)
    {
    }
}
```

以汇编语言编写的相同代码如下。

```
;***************************************************************************
;    EEPROM User Module Example Code:
;
;    A SerialNumber EEPROM was created to start at block 250 with a length
;    of 10 bytes. Remember to edit the Flashsecurity.txt file to allow
;    writes to Flash (Set the 250th block to U - Unprotected).
;
;    This example:
;
;        a) Writes the initial data to the EEPROM block area
;        Note that this will invoke the SavePartial algorithm which
;        allocates a temporary 64-byte buffer on the stack. If
;        Flash memory is plentiful and the extra 54 bytes can be
;        wasted, set the SerialNumber device to a length of 64 and when
;        writing, specify a bytecount of 64. This will write an entire
;        block without using a temporary buffer. The extra 54 bytes
;        beyond the SerialNumber will be bogus data.
;
;        b) Reads the data back into a RAM buffer
;
;***************************************************************************
include "m8c.inc"       ; part specific constants and macros
include "memory.inc"    ; Constants and macros for SMM/LMM and Compiler
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

ADDRESS_OFFSET:             equ     0
NUMBER_OF_BYTES:            equ     10
TEMPERATURE:                equ     25

export _main
export  _SerialNumberExample            ; C export
export   SerialNumberExample            ; assembler export

export  abSerialNumberString
export  abInitialSerialNumber
export  abSerialNumberBuffer
export  bCounter
export  pPtr

    AREA    bss     (RAM,REL)

abInitialSerialNumber:    blk    NUMBER_OF_BYTES   ; string holds initial serial data
abSerialNumberBuffer:     blk    NUMBER_OF_BYTES   ; buffer to read back serial data
bCounter:                 blk     1                ; counter to load initial string
```

```
pPtr:                       blk     1               ; pointer to initial string

    AREA    text (ROM,REL)

;Table to hold initial serial number string
.LITERAL
abSerialNumberString:    db     '0','1','2','3','4','5','6','7','8','9'
.ENDLITERAL

_main:

; load the serialnumber into RAM
    mov    [bCounter], NUMBER_OF_BYTES       ; load 10 bytes from ROM into RAM

    RAM_SETPAGE_MVR >abInitialSerialNumber
    RAM_SETPAGE_MVW >abInitialSerialNumber
    mov    [pPtr], <abInitialSerialNumber ; ptr RAM data to put Flash data
    mov    X, <abSerialNumberString       ; LSB of abSerialNumberString
    mov    A, >abSerialNumberString          ; MSB of abSerialNumberString

; Use ROMX and MVI to copy the data from Flash to RAM
.loop:
    push   A                              ; Save MSB of abSerialNumberString to Stack
    romx
    mvi    [pPtr], A                      ; Save the value to RAM
    pop    A                              ; Get MSB of abSerialNumberString from Stack
    inc    X                              ; Increment LSB of abSerialNumberString
    dec    [bCounter]
    jnz    .loop


; Write the Serial Number - assume temp of 25C
    mov    A, TEMPERATURE                 ; temperature = 25C
    push   A
    mov    A, >NUMBER_OF_BYTES            ; MSB of wByteCount = 0
    push   A
    mov    A, <NUMBER_OF_BYTES        ; LSB of wByteCount = 10
    push   A
    mov    A, >abInitialSerialNumber      ; MSB of pbDest= >abInitialSerialNumber
    push   A
    mov    A, <abInitialSerialNumber  ; LSB of pbDest=abInitialSerialNumber
    push   A
    mov    A, >ADDRESS_OFFSET             ; MSB of wAddr=0
    push   A
    mov    A, <ADDRESS_OFFSET         ; LSB of wAddr=0
    push   A
    call   E2PROM_bE2Write                ; Write the data
    add    SP, -E2_WR_ARG_STACK_FRAME_SIZE
    pop    X


    ;if ( bError == NOERROR )
    cmp    A, 0
    jnz    .ExampleDone
```

```
    ; Read the Serial Number back into a RAM buffer
    mov     A, >NUMBER_OF_BYTES             ; MSB of wByteCount = 0
    push    A
    mov     A, <NUMBER_OF_BYTES           ; LSB of wByteCount = 10
    push    A
    mov     A, >abSerialNumberBuffer        ;MSB of pbDest= >abSerialNumberBuffer
    push    A
    mov     A, <abSerialNumberBuffer      ; LSB of pbDest=abInitialSerialNumber
    push    A
    mov     A, >ADDRESS_OFFSET              ; MSB of wAddr=0
    push    A
    mov     A, <ADDRESS_OFFSET            ; LSB of wAddr=0
    push    A
    call    E2PROM_E2Read
    add     SP, -E2_RD_ARG_STACK_FRAME_SIZE
    pop     X

.ExampleDone:
    jmp .ExampleDone
```